COMPUTER SCIENCES AND SOFTWARE ENGINEERING
AUBURN UNVERSITY

COMP 2710
Software Construction

Summer 2013

Lab 3
The Maze Game

Due: July 24, 2013

Points Possible: 100
Due: Written Portion: July 17th, 2013 by 11:55 pm. turned in via CANVAS SUBMISSION (25 points)
Program: July 24th, 2013 by 11:55 pm turned in via CANVAS SUBMISSION (75 points)

No late assignments will be accepted.
No collaboration between students. Students should NOT share any project code with each other. Collaborations in any form will be treated as a serious violation of the University's academic integrity code.

Goals:

- To develop a maze application that use graph representation
- To learn about how to manipulate graph data structures
- To perform Object-Oriented Analysis, Design, and Testing
- To learn the use of pointers and dynamic memory allocations

Process – 25 points:
Create a text, doc, or .pdf file named "<username>-3p" (for example, mine might read "lim-3p.txt") and provide each of the following. Please submit a text file, a .doc file or .pdf file (if you want to use diagrams or pictures, use .doc or .pdf). You are free to use tools like Visio to help you, but the final output needs to be .txt, .doc, or .pdf.

1. **Analysis:** Prepare use cases. Remember, these use cases describe how the user interacts with a messaging system (what they do, what the systems does in response, etc.). Your use cases should have enough basic details such that someone unfamiliar with the system can have an understanding of what is happening in the maze system interface. They should not include internal technical details that the user is not (and should not) be aware of.

2. **Design**:
   a. Create a Class Diagram (as in Lab 2). Be sure to include:
      1) The name and purpose of the classes
      2) The member variables and the functions of the class

3) Show the interactions between classes (for example, ownership or dependency)
4) Any relevant notes that don't fit into the previous categories can be added

b. Create the data flow diagrams. Show all the entities and processes that comprise the overall system and show how information flows from and to each process.

3. **Testing**: Develop lists of <u>*specific*</u> test cases OR a driver will substitute for this phase:

1) For the system at large. In other words, describe inputs for "nominal" usage. You may need several scenarios. In addition, suggest scenarios for abnormal usage and show what your program should do (for example, entering a negative number for a menu might ask the user to try again).
2) For each object. (Later, these tests can be automated easily using a simple driver function in the object)

4. Implement the maze system

5. **Test Results**: ***After developing the maze game system***, actually try all of your test cases (both system and unit testing). Show the results of your testing (a copy and paste from your program output is fine – don't stress too much about formatting as long as the results are clear). You should have test results for every test case you described. If your system doesn't behave as intended, you should note this. Note: Driver output will substitute for this phase.

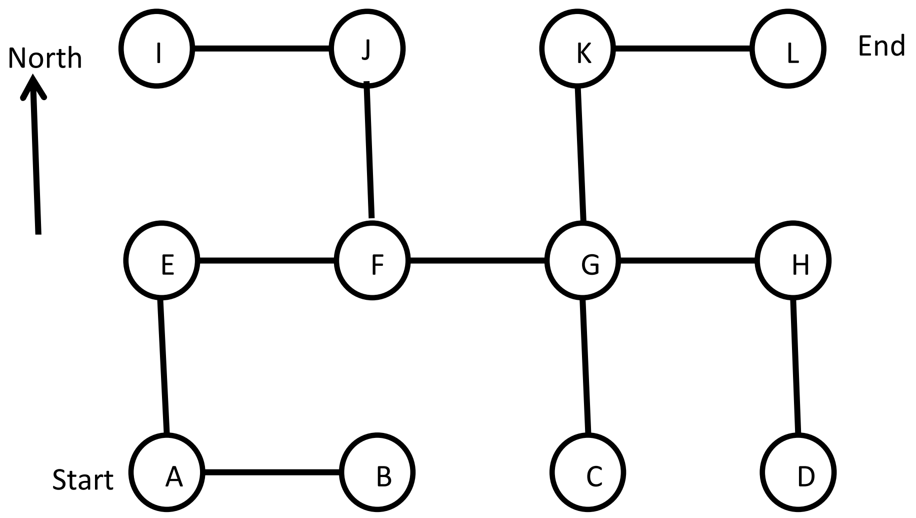
<u>***Program Portion:***</u>

The graph abstraction is an important one since it is used in many different areas of computer sciences and software engineering. For instance, the Internet makes use of graphs representing the Internet router network configuration in order to determine the best route for forwarding packets from the source at one end of the Internet to the destination at the other end. Once a graph is constructed to represent the Internet configuration accurately, then the router executes an algorithm to determine the shortest path from each node to each destination and stores the information in its routing table.

In Lab 3, you will have fun constructing a similar graph to represent a maze which can be used by the user to implement a simple game. You will analyze, design, implement and test the maze game system. Once the graph is constructed by the game system, the user will try to traverse the maze from a starting point to a finish point, by selecting the direction to take at each node. Since the maze configuration is hidden from the user, he/she will first randomly select each direction and learn and remember past dead-end routes and try to find the correct route to the finish point. When the user reaches the finish point, the program will print out the number of steps taken by the user.

What is a graph?

A graph is consists of multiple nodes and edges. For instance, a graph can be represented by the figure below, where the circles represent the nodes and the lines represent the edges. An edge connects two nodes. In this lab assignment, we will only consider special graphs, where each node can have *at most* four edges, in the four directions: North, East, South and West. An implementation detail to note here is that although the edge is represented by a single line, there are actually two links in both directions between the two nodes. For example, the line between A and B below is implemented by a link from A to B and another link from B to A.



What is the maze game system?

In the maze game, there is a maze of rooms and passages connecting the rooms. The maze of rooms and passages can be represented by a graph, such as the graph shown above. The nodes can be thought of as rooms and an edge represents a passage that connects one room to another. Each room can be connected to at most four rooms in the four directions: North, East, South and West, as represented by the graph above.

In this lab assignment, you will implement the maze above using references to instances of a `Node` class, which you will define. Each node in the graph will correspond to an instance of `Node`. The edges correspond to the links that connect one node to another and can be represented in `Node` as instance variables that reference another `Node` class object. Since there are four possible links, `Node` must contain four links (pointers) to other `Node` objects.

The `Start` variable references the node where the user starts, which may represent the room `A`. The goal is to reach the finish which is the node that is referenced by the `Finish` variable which may reference the node representing the room `L`.

Graph Configuration

The configuration of the graph that you will use in the program will be read in from a text file. Each line in the file indicates information on each node, i.e. the name of the node, and the links that may exist from that node to another node in the North, East, South and

West directions. If there is no link in a specific direction, then there is an '*' in place of the node name. For example, the configuration file for the graph above is as follows:

```
A  E  B  *  *
B  *  *  *  A
C  G  *  *  *
D  H  *  *  *
E  *  F  A  *
F  J  G  *  E
G  K  H  C  F
H  *  *  D  G
I  *  J  *  *
J  *  *  F  I
K  *  L  G  *
L  *  *  *  K
```

In the first line of the configuration file above, the name of the node is A. It has a pointer to node E in the North direction and a pointer to node B in the East direction but no link in either the South or West direction. Thus the South and West pointers are set to NULL.

Your program will first read the graph configuration file and construct the graph data structures used for the maze game. Your program must work with any graph configuration file. Several test configuration files will be released to you close to the deadline for you to test the correct execution of your program. *All graph configuration files will have exactly twelve nodes, but their edges will be different.*

The game will traverse the graph based on the inputs given by the user. When the user is at a current room, the program will output the possible moves in the North, East, South or West directions. The user will then enter the desired direction. While the program traverses the graph, it also counts the number of steps. Upon reaching the finish point, it will print out the congratulation banner and the number of steps taken.

The user interface must check for correct input value from the users. If there is any error, e.g. selecting an invalid direction, then the program must display the appropriate error message and continue to prompt for the correct input. Your program must not exit or terminate when there is an incorrect input value.

The name of your program must be called <username>_3.cpp (for example, mine would read "lim_3.cpp"

Use comments to provide a heading at the top of your code containing your name, Auburn Userid, and filename. You will lose points if you do not use the specific program file name, or do not have a comment block on **EVERY** program you hand in.

Your program's output need not exactly match the style of the sample output (see the end of this file for one example of sample output).

Important Notes:
You must use an object-oriented programming strategy in order to design and implement this maze game system (in other words, you will need write class definitions and use those classes, you can't just throw everything in main()). A well-done implementation will produce a number of robust classes, many of which may be useful for future programs in this course and beyond. Some of the classes in your previous lab project may also be re-used here, possibly with some modifications. Remember good design practices discussed in class:

a) A class should do one thing, and do it well
b) Classes should NOT be highly coupled
c) Classes should be highly cohesive

- You should follow standard commenting guidelines.

- You DO NOT need any graphical user interface for this simple, text-based application. If you want to implement a visualization of some sort, then that is extra credit.

Error-Checking:

You should provide enough error-checking that a moderately informed user will not crash your program. This should be discovered through your unit-testing. Your prompts should still inform the user of what is expected of them, even if you have error-checking in place.

Submit your program through the Canvas system online. If for some disastrous reason Canvas goes down, instead e-mail your submission to TA – Sankari Anupindi – at sza0033@tigermail.auburn.edu. Canvas going down is not an excuse for turning in your work late.

You should submit the two files in digital format. No hardcopy is required for the final submission:

<username>_3.cpp
<username>_3p.txt (script of sample normal execution and a script of the results of testing)

A sample execution is shown below, where the bold fonts indicate input by the user.

> *lim_3*

```
        ============================================================
        |              Welcome to the Maze!                |
        ============================================================

You are currently in Room A of the Maze, you can go North or East. What
is your choice? N

You are currently in Room E of the Maze, you can go East or South. What
is your choice? E

You are currently in Room F of the Maze, you can go North or East or
West. What is your choice? E

You are currently in Room G of the Maze, you can go North or East or
South or West. What is your choice? N

You are currently in Room K of the Maze, you can go East or South. What
is your choice? E

Congratulations! You have reached the finish point.
You took 5 steps.
```